DEPARTEMENT TOEGEPASTE
ECONOMISCHE WETENSCHAPPEN

ONDERZOEKSRAPPORT NR 9505

Assembly Line Balancing by Resource-Constrained Project Scheduling Techniques
- A Critical Appraisal -

by

Bert DE REYCK

Willy HERROELEN

Katholieke Universiteit Leuven

Naamsestraat 69, B - 3000 Leuven

ONDERZOEKSRAPPORT NR 9505

Assembly Line Balancing by Resource-Constrained Project Scheduling Techniques
- A Critical Appraisal -

by

Bert DE REYCK

Willy HERROELEN

D/1995/2376/5

ASSEMBLY LINE BALANCING BY RESOURCE-CONSTRAINED PROJECT SCHEDULING TECHNIQUES

- A CRITICAL APPRAISAL -

Bert DE REYCK

Willy HERROELEN

Department of Applied Economics
Katholieke Universiteit Leuven
Hogenheuvel College
Naamsestraat 69, B-3000 Leuven (Belgium)
E-mail: Bert.DeReyck@econ.kuleuven.ac.be
or Willy.Herroelen@econ.kuleuven.ac.be
Phone: 32-16-32 69 66 or 32-16-32 69 70
Fax: 32-16-32 67 32

This research was sponsored by the Belgian National Science Foundation, Research Contract FKFO N° 2.0051.94, which is gratefully acknowledged.

ASSEMBLY LINE BALANCING BY RESOURCE-CONSTRAINED PROJECT SCHEDULING TECHNIQUES - A CRITICAL APPRAISAL

ABSTRACT

Numerous optimal and suboptimal procedures have been developed for solving the simple assembly line balancing problem (SALBP). The similarity between the SALBP and the problem of scheduling project network activities under renewable resource constraints has been noted in many writings. In this paper we exploit the similarity between the SALBP and the generalized resource-constrained project scheduling problem (GRCPSP) by studying two GRCPSP-based formulations for the SALBP. Both formulations assume a single renewable resource. The first formulation assumes finish-start precedence constraints, unit resource requirements and varying resource availability. The second formulation assumes start-start precedence relations, unit activity durations and constant resource availability. The possibility of optimally solving the SALBP is explored by comparing the performance of a branch-and-bound procedure for the GRCPSP for both problem settings with the performance of dedicated SALBP procedures on an extensive set of problem instances. Extensive conclusions are obtained on the potential of various complexity measures for explaining variations in the CPU time required for solving SALBP instances.

1. Introduction

The *simple assembly line balancing problem (SALBP)* involves the grouping of a number of work elements (tasks), each with known performance time, among work stations, each with the same time capacity (cycle time), without violating any precedence relationships between the tasks. This work deals with the type-1 formulation of the SALBP (denoted as *SALBP-1* by Baybars 1986), where tasks have to be grouped so that the number of work stations is minimized. The SALBP-1 is NP-hard because it is a special case of the bin-packing problem, which is known to be NP-hard (Wee and Magazine 1982). A review of line balancing problems can be found in Herroelen (1980). Baybars (1986) offers a survey of exact algorithms while a review of heuristic line balancing techniques can be found in Talbot et al. (1986). Recent computational experience has been reported by Johnson (1993). The *resource-constrained project scheduling problem (RCPSP)* involves the scheduling of project activities subject to zero-lag finish-start precedence relations and constant resource availability constraints in order to minimize the total project duration. The *generalized resource-constrained project scheduling problem (GRCPSP)* extends the RCPSP to the case of precedence diagramming type of precedence constraints, activity ready times and due dates and variable resource availabilities. Both the RCPSP and the GRCPSP are known to be NP-hard (Blazewicz et al. 1983). Reviews of the RCPSP can be found in Herroelen (1972), Davis (1966, 1973) and Icmeli (1993). A survey of the recent advances in optimal procedures for both the RCPSP and the GRCPSP can be found in Herroelen and Demeulemeester (1994).

The strong similarities among the SALBP-1 and the RCPSP have long been recognized by quite a number of authors (Wiest 1963, Wilson 1964, Mandeville 1965, Moodie and Mandeville 1966, Drexl 1990). Sprecher (1994) seems to be the only author to formulate the SALBP-1 as a GRCPSP with unit resource requirements and varying resource availabilities. These strong similarities suggest the cross-application of solution procedures which is considered as an important development in the field by Moodie and Mandeville (1966), Davis (1973) and Drexl (1990). In this paper we present an additional GRCPSP-type of formulation for the SALBP-1 which assumes unit task durations, start-start precedence relations and constant resource availabilities. We then use both GRCPSP based formulations to explore the possibilities for optimally solving the SALBP-1 using an optimal procedure for solving the GRCPSP. Results are obtained on a set of test problems and contrasted to the results obtained using dedicated SALBP-1 algorithms.

The organization of the paper is as follows. In §2 we present a mathematical programming formulation of the SALBP-1 problem. In §3 we present a mathematical programming formulation of the RCPSP. The relationship between the SALBP-1 and the

RCPSP, as established in the literature, is reviewed in §4. In §5 we describe the GRCPSP formulation of Sprecher (1994) and present a new GRCPSP formulation based on unit task durations and start-start precedence relationships. Computational results are obtained in §6, while §7 is reserved for our overall conclusions.

2. The SALBP-1 Problem

The simple assembly line balancing problem (SALBP) is concerned with the grouping of a number of work elements (tasks) among a number of workstations on an assembly line. The following assumptions hold (Baybars 1986):

- all input parameters are known with certainty;
- a task cannot be split among two or more stations;
- tasks cannot be processed in arbitrary sequences due to technological precedence constraints;
- all tasks must be processed;
- all stations under consideration are equipped and manned to process any of the tasks;
- task process times are fixed and not sequence dependent;
- any task can be performed at any station;
- the line is considered to be serial (process times are additive at any station);
- the serial assembly line is designed for a single model.

SALBP-1, the first version of SALBP, adds the assumption that the cycle time is fixed, while SALBP-2, the second version, assumes that the number of stations is given and fixed. This paper only addresses SALBP-1. The objective is to minimize total slack, which is equivalent to minimizing the number of stations along the line (in SALBP-2, the objective is to minimize the cycle time, or equivalently, to maximize the production rate).

Many alternative mathematical programming formulations have been presented in the literature (Salveson 1955, Bowman 1960, White 1961, Thangavelu and Shetty 1971, Patterson and Albracht 1975, Talbot and Patterson 1984). We present our formulation based on the following notation:

c = cycle time

t_k = processing time of task k, $k = 1, 2, ..., K$

$S_k(P_k)$ = subset of tasks which must succeed (precede) task k, $k = 1, 2, ..., K$

W_i = subset of tasks which could be assigned to station i, $i = 1,2,..., M$

$F = \{k \mid S_k = \varnothing\}$, set of tasks without successors

d = dummy end node; $t_d = 0$ and $k \prec d$, $\forall\, k \in F$.

M = upper bound on the number of work stations

$$x_{ki} = \begin{cases} 1, & \text{if work element } k \text{ is assigned to station } i \\ 0, & \text{otherwise} \end{cases}$$

Patterson and Albracht (1975) define for each task k, the earliest station E_k and the latest station L_k that task can be assigned to, based upon the precedence relations:

$$E_k = \begin{cases} 1, & \text{for } t_k + \sum_{j \in P_k} t_j = 0, \quad k = 1,2,\ldots,K \\ \left\lceil (t_k + \sum_{j \in P_k} t_j)/c \right\rceil^+, & \text{otherwise} \end{cases}$$

$$L_k = \begin{cases} M, & \text{for } t_k + \sum_{j \in S_k} t_j = 0, \quad k = 1,2,\ldots,K \\ M + 1 - \left\lceil (t_k + \sum_{j \in S_k} t_j)/c \right\rceil^+, & \text{otherwise} \end{cases}$$

where the notation $\lceil a \rceil^+$ denotes the smallest number greater than or equal to a. The alternative definitions 1 (M) for E_k (L_k) are needed in case a dummy node is used in the precedence diagram preceding (succeeding) all other nodes.

The formulation can then be written as follows:

$$\text{Minimize} \sum_{i=E_d}^{M} i x_{di} \qquad [1]$$

Subject to

$$\sum_{i=E_k}^{L_k} x_{ki} = 1, \qquad k = 1,2,\ldots,K \qquad [2]$$

$$\sum_{i=E_a}^{L_a} i x_{ai} \le \sum_{j=E_b}^{L_b} j x_{bj}, \qquad \text{where } a \prec b \qquad [3]$$

$$\sum_{k \in W_i} t_k x_{ki} \le c, \qquad i = 1,2,\ldots M \qquad [4]$$

$$x_{ki} \in \{0,1\} \qquad k = 1,2,\ldots,K; \quad i = 1,2,\ldots,M \qquad [5]$$

In the objective function Eq. [1], E_d denotes the earliest station in which task d (the dummy end node) can be assigned. Minimizing the expression is equivalent to minimizing the number of work stations. Eqs. [2] denote that each task must be assigned to only one work station. Eqs. [3] denote the precedence constraints. Note that we have assumed that the tasks assigned to a particular work station have to be performed consecutively. Eqs. [4] represent the cycle time constraints. Eqs. [5] force the decision variables to assume binary values.

3. The RCPSP

The resource-constrained project scheduling problem (RCPSP) is concerned with the scheduling of project activities subject to finish-start precedence relations and availability constraints on one or more renewable resources. The following assumptions hold:

- the project is represented by an acyclic activity-on-node (AoN) network with single start node 1 and single end node N;
- all input parameters are known with certainty;
- an activity cannot be split;
- activity process times are fixed;
- activities are subject to finish-start precedence constraints with a zero time lag;
- renewable resources are available in constant amounts over time.

The objective is to minimize the project duration. Many alternative mathematical programming formulations have been presented in the literature (Bowman 1959, Pritsker et al. 1969, Elmaghraby 1977). We present our formulation based on the following notation:

d_j = processing time of activity j; j=1,2,...,N

P = set of all precedence related activities

F = set of activities without successors

r_{js} = resource requirement of resource type s by activity j, j=1,2,...,N; s=1,2,...,S

A_s = availability of resource type s

HP = upper bound on the project duration

J_t = set of activities which are active in period t, t=1,2,...,HP

$$x_{jt} = \begin{cases} 1, & \text{if activity } j \text{ starts in period } t \\ 0, & \text{otherwise} \end{cases}$$

Elmaghraby (1977) shows how the precedence relations allow to derive the earliest starting period E_j and latest allowable starting period L_j for each activity j. The formulation can then be written as follows:

Minimize Z [6]

Subject to

$$\sum_{t=E_j}^{L_j} x_{jt} = 1, \qquad j = 1,2,\ldots,N \qquad [7]$$

$$\sum_{t=E_p}^{L_p} t x_{pt} + d_p \le \sum_{t=E_q}^{L_q} t x_{qt}, \qquad \forall (p,q) \in P \qquad [8]$$

$$\sum_{j \in J_t} \sum_{u=E_j}^{L_j} r_{js} x_{ju} \le A_s, \qquad s = 1,2,\ldots,S; \quad t = 1,2,\ldots,HP \qquad [9]$$

$$\sum_{t=E_j}^{L_j} t x_{jt} + d_j \le Z, \qquad j \in F \qquad [10]$$

$$x_{jt} \in \{0,1\} \qquad j = 1,2,\ldots,N; \quad t = 1,2,\ldots,HP \qquad [11]$$

Z : integer variable with upper bound $HP + 1$ [12]

The objective function given in Eq. [6] is to minimize the project duration. As specified in Eq. [12], Z is an integer decision variable with upper bound HP+1. Eqs. [7] represent the activity occurrence constraints. The activity precedence constraints are denoted by Eqs. [8]. The resource constraints are denoted in Eqs. [9]. The project completion constraints are specified in Eqs. [10]. Eqs. [11] force the decision variables to assume binary values.

4. Similarities among SALBP-1 and RCPSP

As Wiest (1963) has pointed out, the SALBP-1 problem differs considerably from the RCPSP, *from a managerial point of view*. The SALBP-1 problem is concerned with repetitive operations and large numbers of identical products, while the RCPSP is mostly a one-time, "one-of-a-kind" operation. On the other hand, the following analogies do exist (Wilson 1964, Mandeville 1965, Davis 1973):

SALBP-1	RCPSP
Work elements (tasks)	Activities
Work element (task) times	Activity resource requirements
Work stations	Days
Cycle time	Maximum available units of resource

Moodie and Mandeville (1966) state that "with these similarities noted it becomes readily apparent that the two problems are about identical. The only basic difference is that an activity in SALBP-1 occurs at one and only one work station, while in the RCPSP an activity can be spread over several days (stations)". Davis (1973) states that the relationships shown above can only be carried to certain limits: "The analogy between cycle

time and resource limits holds strictly only in the case of one resource type for the RCPSP. Also the analogy between stations and days is tenuous since in the SALBP-1 problem tasks in series may be performed at the same station, whereas serial activities in the RCPSP are never performed on the same day. The latter difference is minimized, however, if activities of the project are presented as a series of unit-duration tasks and performance constraints similar to zoning constraints of the line balancing problem are imposed." Elmaghraby (1977) argues along the following lines: "It is easy to see that the line balancing problem is identical to the scarce-resources problem in activity networks with the following structure: (a) there is only one scarce resource, (b) each activity is of duration one period, (c) the task time represents the quantity of the resource required by the corresponding activity in that period, and (d) the cycle time represents the resource availability. A 'station' now corresponds to a period, and minimizing the number of stations is equivalent to minimizing the project duration".

The crossover from SALBP-1 to RCPSP was accomplished by Black (1965) who developed a resource levelling algorithm based on a modification of Gutjahr and Nemhauser's (1964) SALBP-1 approach. Gutjahr and Nemhauser begin by generating a directed network (the *A*-network) based on the precedence diagram the nodes of which correspond to feasible subsets, with the source node being the null set and the sink node being the set of all tasks. A feasible subset is a subset of tasks that can be processed without prior completion of any other task and in any order that satisfies the precedence relations. The time τ_i associated with a node i is simply the sum of the processing times of the tasks represented by that node. Let Q(i) denote the subset of tasks represented by node *i*. An arc (i,j) is in the A-network iff Q(i) is a subset of Q(j) and $\tau_i - \tau_j \leq c$. Thus, an arc (i,j) corresponds to a station assignment of all tasks in Q(j) but not in Q(i). As a result, there is a one-to-one correspondence between source-sink paths and feasible assignments. Finding the shortest path in the A-network is equivalent to finding the minimum number of stations required. The crossover is established by splitting the network activities in unit duration tasks with a must follow precedence relation and replacing the cycle time by the single resource availability. The basic problem with this crossover is the size of the A-network. Consequently (see Baybars (1986)), the method is not deemed practical. Davis and Heidorn (1971) used this approach to solve the RCPSP optimally, by transforming the RCPSP with a single resource type to a SALBP (by splitting up the activities in unit duration activities).

Drexl (1990) discusses how the SALBP-1 can be transformed into a RCPSP by making the following assumptions:

- the tasks of the SALBP-1 correspond to the activities in the RCPSP;

- the task times in the SALBP-1 correspond to the processing times of the activities in the RCPSP;
- the precedence relations of the SALBP-1 correspond to the precedence relations among the activities in the RCPSP;
- a single resource in the RCPSP has a constant availability equal to the cycle time;
- each activity j uses d_j units of the renewable resource each period it is in progress, where d_j denotes the duration of activity j (which is equal to the duration of task j);
- the objective is the minimization of the makespan.

The author offers a numerical example illustrating the relationship between the SALBP-1 and the RCPSP, which still leaves a number of open problems (Drexl 1994) which have been subsequently removed by Sprecher (1994). The formulation developed by Sprecher (1994) is presented in the next section. There we also present a new formulation based on unit task durations and start-start precedence relations.

5. Formulating the SALBP-1 as a GRCPSP

5.1. The model by Sprecher

Sprecher (1994) reformulates the SALBP-1 problem as a *generalized resource-constrained project scheduling problem (GRCPSP)* using the following assumptions:

- the tasks correspond to the activities;
- the task times correspond to the activity durations;
- the precedence relations between the activities are the ones induced by the one-to-one correspondence of the activities and the tasks;
- HP is defined as $M(c+1)$ and a renewable resource is introduced with an availability of

$$A_t = \begin{cases} 0, & \text{if } t = i(c+1), \quad i = 1,2,\ldots,M \\ 1, & \text{otherwise} \end{cases} \qquad \text{in period } t = 1,2,\ldots,HP.$$

- each activity uses one unit of the renewable resource each period it is in process;
- the objective is the minimization of the makespan.

The essential clue in the Sprecher formulation is the *time-varying supply* of the single renewable resource. Since no unit of the resource is available in the periods t, $t = i(c+1)$, $i = 1,2,\ldots,M$, and , moreover, each activity j, $j = 1,2,\ldots,N$, uses one unit of the resource, it has to be performed in an interval $[(i-1)(c+1)+1, i(c+1)-1]$, $i = 1,2,\ldots,M$. From an optimal solution of the GRCPSP, one can derive the optimal solution of the assembly line balancing problem. The activities are assigned to the stations related to the intervals in which the activities are performed. Consider the example given in Figure 1 (Jackson 1956). The numbers inside the nodes denote the task numbers, the ones above each node denote the task times. The optimal schedule corresponding to the GRCPSP-formulation of

Sprecher is represented by its resource profile in Figure 2, which also shows the time-varying supply of the renewable resource by the dotted lines. The optimal workstation assignment can easily be found by relating the intervals with the corresponding stations.

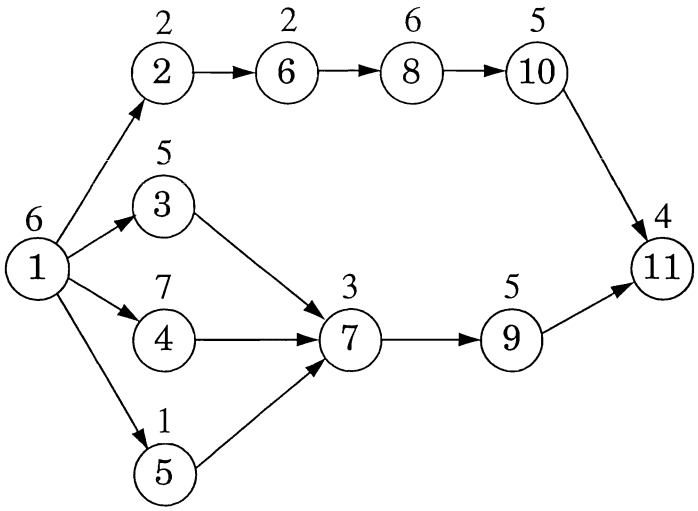

Figure 1 : The 11-task problem (Jackson 1956)

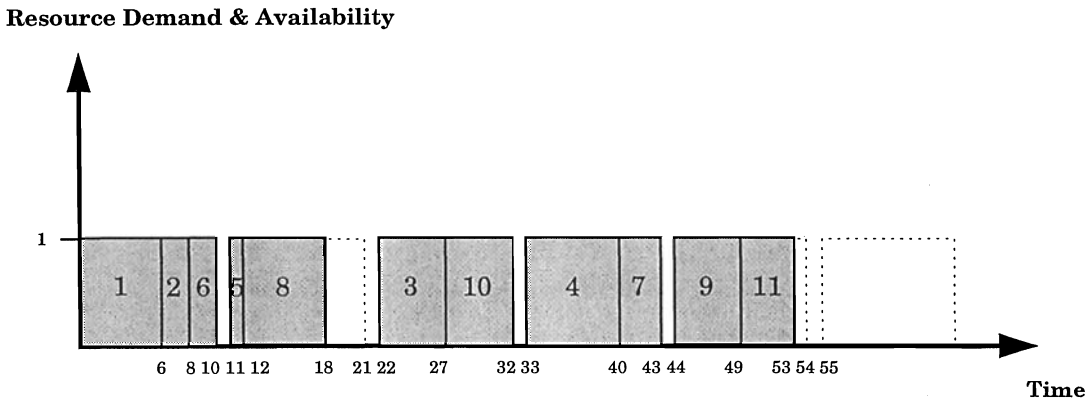

Figure 2 : Resource profile of the optimal solution of the Sprecher GRCPSP formulation

5.2. A new formulation

While the Sprecher formulation assumes unit resource requirements and time-varying resource availability, an alternative formulation can be based on unit activity durations, constant resource availability and start-start precedence relations between the activities. The formulation is based on the following assumptions:

- the tasks correspond to the activities;
- the activities have a unit duration;
- the activities are subject to start-start precedence relations with a time lag of zero;

- each activity requires the single renewable resource in an amount equal to the processing time of the corresponding task;
- the availability of the single renewable resource is constant over time and equal to the cycle time;
- the objective is to minimize the makespan.

Since activities have unit task durations each day in the schedule corresponds to a work station. The daily constant resource availability constraint corresponds to the cycle time constraint. The start-start precedence relations allow a task to be assigned to a work station as soon as its predecessor task has been started at the same or at one of the preceding work stations. Minimizing makespan corresponds to minimizing the number of work stations. Consider the same example given in Figure 1. The optimal schedule corresponding to the new GRCPSP-formulation is represented by its resource profile in Figure 3. The optimal workstation assignment can easily be found by relating the time periods with the corresponding stations.

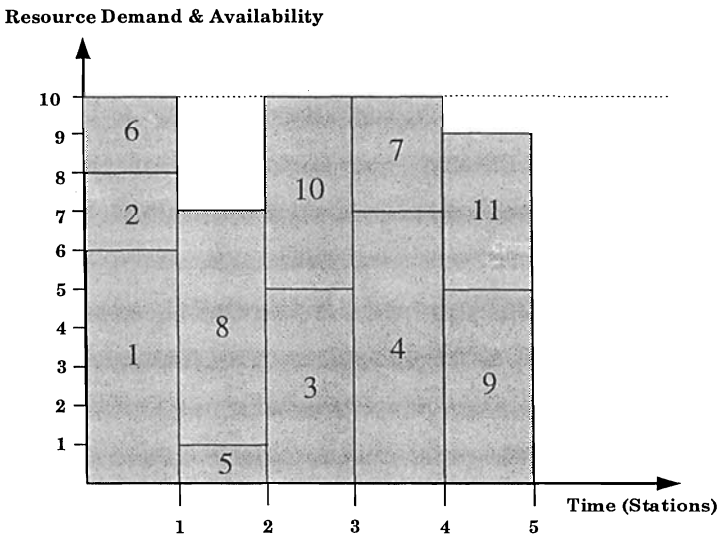

Figure 3 : Resource profile of the optimal solution of the New GRCPSP formulation

5.3 A common solution procedure

Recently, an optimal branch-and-bound procedure (denoted as the *GDH-procedure*) has been presented by Demeulemeester and Herroelen (1994) for solving the GRCPSP. In the GRCPSP, three of the assumptions of the RCPSP are relaxed. First of all, the GRCPSP allows for other than finish-start precedence relations (start-start, finish-finish, finish-start, and start-finish) with the only restriction that activities are not allowed to start before one of their predecessors has started. Secondly, ready times and deadlines may be specified for each activity stating that the activity cannot be started earlier than its ready time and must be completed by its deadline. Finally, the resource availabilities may be

variable over the project duration. Computational experience with the GDH-algorithm on a personal computer has been promising. Given its underlying assumptions, the GDH-procedure can be used for solving SALBP-1 problems based on the two formulations presented in the two previous sections.

6. Computational experiment

The objective of this section is to examine the potential of the two formulations (subsequently denoted as *Sprecher* and *New*) given in the preceding section for solving the SALBP-1 problem. The GDH-procedure will be used as the vehicle of analysis. The performance of the GDH-procedure will be contrasted to the performance of three dedicated procedures for solving the SALBP-1. The first dedicated procedure used in our experiment is the branch-and-bound procedure developed by Van Assche and Herroelen (1978). Given its intrinsic 'simplicity', this procedure (subsequently denoted as *VH*) can serve as an excellent benchmark. In addition, the procedure has been recently applied to the problem of balancing U-shaped lines (Sparling and Miltenburg 1994). Since its development in 1978, VH has undergone some changes in order to enhance its efficiency :

- An upper bound is derived from the number of stations which results from assigning tasks to stations in the order in which they appear in the so-called topological ranking list (Spaas & Herroelen 1992). A task only enters such an ordered list as soon as all its predecessors have already been listed. It should be clear that many such lists may be created for the same precedence network.
- The original branching rule of the VH procedure is to branch from the node with the current smallest lower bound, where ties are resolved by a number of tie-breaking rules. The tie-breaking rule in the new implementation is to branch from the deepest node in the search tree, using minimum cumulative slack as a second tie-breaker.
- The VH procedure has been recoded in Visual C++ (the original algorithm was coded in Fortran for use on a mainframe), and compiled on a 486sx computer running at 33Mhz (without coprocessor) under Windows NT (32-bit).

The other dedicated procedures used in our experiment are FABLE (Johnson 1988), a 'laser'-type depth-first branch-and-bound procedure which seems to be able to solve very large SALBP instances (Johnson 1988), and EUREKA (Hoffmann 1992), a very simple, but elegant, branch-and-bound algorithm combined with a heuristic. FABLE and EUREKA are considered to rank among the best algorithms currently available. Both algorithms were originally coded in Fortran and perform best when used with a co-processor. In the absence of a 32-bit compiler, we compiled both codes using the Microsoft compiler version 5.1 to run on a 486dx computer (with co-processor) running at 33 Mhz under DOS (16-bit).

Two sets of test problems have been used in our experiment. The first set consists of a number of test problems taken from the literature. A second battery of test problems was generated using the network generator PROGEN developed by Kolisch et al. (1992).

6.1. Literature problem set

It is common practice to validate optimal and heuristic solution procedures for the SALBP on the standard set of 64 test problems, assembled by Talbot and Patterson (1984) from the literature (Johnson 1988, 1993; Hoffmann 1992). We tested the two GRCPSP formulations for the SALBP on a subset of these problems, covering only the smaller problem instances. The results of this analysis can be found in Table 1, which lists for each problem, the reference, the number of tasks, the cycle time, the optimal number of stations and the CPU times (in seconds) obtained by each solution procedure. For EUREKA, the time window L is *arbitrarily* set at 0.1 seconds. A dash indicates that the problem could not be solved to optimality within the time limit of 3600 seconds.

Table 1 : SALBP CPU times for the literature problem set

Author	Tasks	Cycle Time	Optimum	VH	FABLE	EUREKA	NEW	SPRECHER
Bowman	9	20	5	.01	.00	.00	.01	.02
Jackson	11	7	8	.01	.00	.00	.02	.10
		9	6	.01	.00	.00	.01	.11
		10	5	.01	.00	.00	.01	.08
		13	4	.01	.05	.00	.02	.05
		14	4	.01	.00	.00	.01	.06
		21	3	.01	.00	.00	.02	.03
Mitchell	21	14	8	.05	.00	.05	.06	14.43
		15	8	.02	.00	.00	.14	10.32
		21	5	.04	.00	.00	.11	9.02
		26	5	.00	.00	.00	.06	2.74
		35	3	.12	.00	.00	.09	5.44
		39	3	.01	.00	.00	.12	1.66
Sawyer	30	25	14	.13	.06	.06	—	—
		27	13	.71	.06	.10	—	—
		30	12	7.22	.77	.20	2346.53	—
		36	10	1.40	.22	.10	—	—
		41	8	.20	.00	.15	—	—
		54	7	3.47	.33	.10	—	—
		75	5	1.00	.05	.00	405.76	—

It immediately appears from this table that the best results are obtained using dedicated line balancing procedures. Although the new GRCPSP formulation is much more efficient than the Sprecher formulation, the corresponding procedures are outperformed by the three dedicated algorithms. Although the similarity between the SALBP and the

GRCPSP is interesting from the conceptual point of view, it seems that both GRCPSP formulations cannot exploit it to their advantage. FABLE clearly outperforms VH, but both procedures seem to be outperformed by EUREKA. This result is consistent with the results obtained by Hoffmann (1992), who compared EUREKA and FABLE on the complete set of 64 problems assembled by Talbot and Patterson (1984).

Essentially, EUREKA is a heuristic procedure. After spending L seconds on forward balancing and again L seconds on reverse balancing, a heuristic is used to obtain a fast, but not necessarily optimal, solution. This time window is critical to the effecitivenss and efficiency of EUREKA, since it determines the trade-off between a higher solution time on one hand and a larger number of suboptimally solved problems on the other. In our experiment, the time window was *arbitrarily* set at 0.1 seconds. This setting allows EUREKA to solve all attempted problems to optimality. In this case, increasing L would yield higher CPU times, without improving effectiveness. Although in general (Hoffmann 1992), the effectiveness of EUREKA does not seem to be very sensitive to changes in L (increasing L will not significantly increase the number of problems solved to optimality), setting 'good' L values remains a tactical problem.

6.2. PROGEN problem set

In order to obtain a more structured insight in the performance of the proposed procedures, we used PROGEN (Kolisch et al. 1992) to generate a total of 6000 networks, while varying a number of parameters which are considered to be important indicators of the complexity of the SALBP (Elmaghraby and Herroelen 1980).

6.2.1. The test problems

These parameters are :

- the *number of tasks* (set at 10, 15, 20, 25, 30)
- the *coefficient of network complexity (CNC)* (Pascoe 1966, Davies 1974), defined as the number of precedence relations divided by the number of tasks. This parameter is often called *'arcs / nodes'* (in our experiment the CNC is set to 1; 1.25; 1.5; 1.75; 2).
- the *order strength (OS)* (Mastor 1970), defined as the number of precedence relations, including redundant ones, divided by $n(n\text{-}1)/2$, where n denotes the number of tasks. This measure is similar to the *flexibility ratio*, defined by Dar-El (1973) as the number of zero entries in the precedence matrix divided by the total number of matrix entries. As Elmaghraby and Herroelen (1980) have observed, the order strength is precisely one minus the flexibility ratio. The order strength is sometimes described as the *density* (Kao and Queyranne 1982). In our experiment, OS varies from 0.4 to 1.

- the *number of feasible sequences (F_{seq})*. The calculation of the exact number of feasible sequences (Ignall 1965) is very hard. An upper bound, F'_{seq}, can be readily calculated as $n!/2^r$ (Ignall 1965), where n denotes the number of tasks and r denotes the number of precedence relations (not including redundant ones). In our experiment, F'_{seq} varies from 20 to 2.47 E+23.
- the *number of feasible subsets (F_{sub})* (Held et al. 1963), which varies in our experiment from 15 to 100,000. The calculation of the number of feasible subsets is NP-hard (Provan and Ball 1983), which makes the usefulness of this parameter as a measure of complexity questionable. Schrage and Baker (1978), however, have developed an efficient labelling scheme that can be used in the SALBP to assign a label to each task. The sum of the labels is an upper bound, F'_{sub}, on the number of feasible subsets. Both F_{sub} and F'_{sub} are used in our experiment.
- the *complexity index (CI)* (De Reyck and Herroelen 1995). Recently, Bein et al. (1992) introduced a new characterization of two-terminal acyclic networks which essentially measures how nearly series-parallel a network is. They define the *reduction complexity* as the minimum number of node reductions sufficient (along with series and parallel reductions) to reduce a two-terminal directed acyclic network to a single edge. We adopt the reduction complexity as our definition of the complexity index (CI) of a network, and include the CI as a new measure of network complexity for the SALBP. Note that the CI is only defined on activity-on-the-arc (AoA) networks, whereas the precedence diagram in a SALBP is usually given in activity-on-the-node (AoN) format. A brief description of the concepts underlying the CI is given in Appendix 1. In our experiment, CI varies from 0 to 19.
- the *average number of tasks per station* in the optimal solution (set at 2, 4, 6), which can be controlled by varying the cycle time (set at 10, 20, 30).

Basically, the measures discussed above can be classified in three broad categories :

- *problem size* parameters, e.g. the number of tasks;
- *network topology* parameters, e.g. the coefficient of network complexity (arcs/nodes) or the order strength;
- combined *problem size* and *network topology* measures, e.g. the number of feasible subsets (F_{sub} and F'_{sub}) and feasible sequences (F_{seq} and F'_{seq});
- *resource* parameters, e.g. the average number of tasks per station

It is obvious that the number of tasks is a very good measure of problem size. Also the average number of tasks per station (which can be varied by varying the cycle time) probably is a very good measure of the 'resource-constrainedness' of the problem. In the assembly line balancing literature, the order strength is widely advocated as a good

measure of network complexity for the SALBP. Baybars (1986), for instance, states that "in general the CPU time for solving a SALB problem increases as its order strength decreases, and, therefore, the order strength is a good predictor of CPU times" (see also Johnson 1981). Although the second statement may be an overstatement, there is no doubt that the order strength, despite its simplicity, is a fairly good measure of the underlying *network* complexity for the SALBP. The same author, however, concludes that there still is a need for good measures of complexity of SALB problems. A good measure of complexity (or a composite set of measures) would greatly facilitate the choice between two proposed algorithms or the determination of the efficiency of a particular algorithm. It would eliminate any possible bias in the conclusions regarding the efficiency of a particular algorithm relative to others by ensuring that the algorithm is evaluated at several points in the range of 'complexity' (Elmaghraby and Herroelen 1980). This inspired us to test the potential of F'_{seq}, F_{sub}, F'_{sub} and CI as precedence network complexity parameters. Note that although the number of feasible subsets and feasible sequences are probably good measures of SALBP complexity, they are not pure measures of *precedence network* complexity, since they are affected by the size of the problem.

6.2.2. Computational results

As mentioned above, we used PROGEN (Kolisch et al. 1992) to generate 60 sets of 100 networks each, using the following parameter settings. The number of tasks is set to 10, 15, 20, 25 and 30. The coefficient of network complexity (arcs/nodes) is set to 1, 1.25, 1.5, 1.75 and 2 (not all combinations of the number of tasks and the coefficient of network complexity could be generated by PROGEN, which explains why we only have 20 and not 25 problem sets). The maximum number of successors resp. predecessors is set to 3. The duration of each task is drawn from the uniform distribution in the range [1,10]. The cycle time is set to 10, 20 and 30, which results in an average of 2, 4 and 6 tasks per workstation. This results in a total of 6000 networks. Each network is solved using the GDH procedure for both GRCPSP formulations (*Sprecher* and *New*), the VH procedure, FABLE and EUREKA. As mentioned before, both the GDH and the VH procedure are coded in Visual C++ and run on a 486sx 33 Mhz computer (without coprocessor) under Windows NT (32-bit). FABLE and EUREKA are coded in Fortran and run on a 486dx 33 Mhz computer (with coprocessor) under DOS (16-bit).

The results are shown in Tables 2 to 7. In each of the tables, a 'cell' relates to the results obtained for a particular problem class obtained by combining a particular value of the cycle time, the number of tasks, the coefficient of network complexity (Table 2), the order strength (Table 3), the complexity index (Table 4), the number of feasible subsets (F_{sub}; Table 5), the estimated number of feasible subsets (F'_{sub}; Table 6) and the estimated

number of feasible sequences (F'_{seq}; Table 7). Except where empty or holding a dash (-), the cell entries consist of five numbers, representing the average CPU time in seconds obtained by VH, FABLE, EUREKA, NEW GRCPSP and SPRECHER GRCPSP on the 100 networks in the corresponding problem class. Blank cells indicate that no such problems could be generated by PROGEN. A dash indicates that no attempt was made to solve the corresponding problems, as obtaining optimal solutions within the allowable time (3600 seconds) was deemed very unlikely.

As mentioned above, the value of the time window L is critical to the performance of EUREKA. The time window L is *arbitrarily* set at 10 seconds. However, as confirmed by Hoffmann (1992), the effectiveness of EUREKA does not seem to be very sensitive to changes in L. For instance, if L is set at 60 seconds instead of 10 seconds, we found that the number of problems not solved to optimality decreases from 276 to 201, whereas the average CPU time increases from 1.66 seconds to 7.66 seconds. The numbers between brackets indicate the number of problems not solved to (proven) optimality.

Table 2 : SALBP CPU times with CNC as network complexity measure

cycle time		10					20					30				
	tasks	10	15	20	25	30	10	15	20	25	30	10	15	20	25	30
CNC																
	VH	.00					.01					.01				
	FABLE	.00					.01					.01				
1	EUREKA	.01					.00					.00				
	NEW	.01					.01					.01				
	SPRECH.	.02					.01					.02				
	VH	.01	.03	.19	.88	20.00	.01	.04	.15	.35	2.30	.01	.09	.67	2.65	24.96
	FABLE	.01	.02	.03	.09	22.91	.00	.01	.02	.02	.10	.00	.01	.02	.02	.02
1.25	EUREKA	.02	1.88 (1)	5.70 (12)	8.00 (27)	10.41 (37)	.00	.04	.10	1.10 (5)	2.38 (11)	.00	.01	.00	.47 (2)	1.23 (6)
	NEW	.01	.05	.49	376.61	—	.01	.06	.70	205.08	—	.01	.10	1.30	32.81	—
	SPRECH.	.06	2.32	2606.57	—	—	.03	.36	796.63	—	—	.02	.22	106.08	—	—
	VH	.01	.02	.10	.61	4.59	.01	.02	.09	.23	.66	.01	.05	.28	1.12	5.60
	FABLE	.01	.02	.02	.06	1.92	.01	.01	.01	.02	.03	.00	.01	.02	.02	.02
1.5	EUREKA	.02	.97	3.14 (6)	6.91 (17)	10.29 (35)	.00	.01	.09	.58 (1)	.97 (3)	.00	.00	.00	.16	.61 (3)
	NEW	.01	.03	.29	7.87	2065.48	.01	.04	.39	12.13	2809.00	.01	.08	.78	8.36	1775.07
	SPRECH.	.04	.53	1197.55	3600.00	—	.02	.18	114.16	3225.16	—	.01	.11	2.40	2584.39	—
	VH	.01	.02	.06	.20	1.31	.01	.02	.05	.10	.32	.01	.03	.10	.33	1.36
	FABLE	.01	.01	.02	.03	.26	.01	.01	.02	.02	.03	.01	.02	.02	.02	.03
1.75	EUREKA	.03	.86	3.18 (2)	5.71 (13)	9.41 (29)	.00	.00	.12	.13	1.07 (4)	.00	.00	.00	.11	.79 (3)
	NEW	.01	.03	.15	1.94	700.13	.01	.03	.17	1.77	868.35	.01	.08	.39	3.34	168,34
	SPRECH.	.07	.32	81.63	3181.67	—	.03	.12	1.98	1656.56	—	.02	.07	.92	492.28	—
	VH		.03	.05	.16	.72		.03	.03	.09	.18		.06	.06	.15	.48
	FABLE		.01	.02	.03	.45		.01	.02	.02	.03		.01	.02	.02	.03
2	EUREKA		1.82	2.83 (2)	6.19 (16)	9.76 (28)		.01	.02	.11	1.33 (6)		.00	.00	.01	.83 (3)
	NEW		.04	.10	1.02	39.76		.05	.12	.86	19.36		.09	.33	2.13	23.68
	SPRECH.		.97	6.71	2412.28	—		.32	1.31	823.65	—		.18	.56	335.74	—

Table 3 : SALBP CPU times with OS as network complexity measure

cycle time		10					20					30				
	tasks	10	15	20	25	30	10	15	20	25	30	10	15	20	25	30
OS																
	VH				.59	22.17				.44	2.60				3.97	31.80
	FABLE				.10	22.88				.01	.03				.01	.02
0.4 - 0.5	EUREKA				7.01	9.33				1.03	1.47				1.54	.75
	NEW				1501.28	3513.31				267.19	3600.00				63.18	3504.58
	SPRECH.				—	—				—	—				—	—
	VH			.24	.90	4.32			.19	.31	.63			.85	2.12	3.23
	FABLE			.03	.10	3.83			.02	.02	.07			.02	.02	.02
0.5 - 0.6	EUREKA			5.26	7.98	10.59			.08	.93	1.49			.00	.19	1.03
	NEW			.63	131.93	1898.51			1.08	127.74	2596.85			1.55	21.42	1656.46
	SPRECH.			3003.10	3600.00	—			1204.79	3576.27	—			142.98	3494.71	—
	VH		.03	.09	.24	.92		.04	.08	.12	.20		.09	.24	.36	.63
	FABLE		.01	.02	.03	.36		.01	.01	.02	.03		.01	.01	.02	.03
0.6 - 0.7	EUREKA		1.46	3.70	6.06	9.68		.03	.15	.31	1.44		.00	.00	.15	.80
	NEW		.05	.24	1.79	164.80		.06	.63	2.04	146.46		.10	.70	3.84	35.22
	SPRECH.		1.45	1078.40	3325.49	—		.41	15.09	1914.57	—		.24	2.22	824.32	—
	VH	.01	.02	.05	.12	.80	.01	.02	.04	.08	.14	.01	.04	.07	.14	.29
	FABLE	.01	.01	.02	.03	.10	.01	.01	.02	.02	.03	.01	.01	.02	.02	.02
0.7 - 0.8	EUREKA	.02	1.36	3.11	5.65	10.25	.00	.01	.03	.06	.00	.00	.00	.00	.03	.00
	NEW	.01	.03	.11	.77	18.61	.01	.04	.15	.66	7.73	.01	.08	.34	1.70	9.64
	SPRECH.	.07	.52	6.71	1940.74	—	.03	.18	1.41	485.08	—	.02	.11	.64	165.88	—
	VH	.01	.02	.03			.01	.01	.02			.01	.02	.04		
	FABLE	.01	.02	.02			.01	.02	.01			.01	.02	.03		
0.8 - 0.9	EUREKA	.01	.53	1.80			.00	.01	.04			.00	.00	.00		
	NEW	.01	.02	05			.01	.03	.07			.01	.06	.15		
	SPRECH.	.03	.16	2.83			.02	.06	.66			.01	.04	.27		
	VH	.00					.01					.00				
	FABLE	.00					.00					.01				
0.9 - 1.0	EUREKA	.01					.00					.00				
	NEW	.01					.01					.02				
	SPRECH.	.01					.01					.01				

The average CPU times of the five procedures for all 6000 problems are :

- VH : 1.20 seconds
- FABLE : 0.44 seconds
- EUREKA : 1.66 seconds (276 problems not solved to optimality)
- NEW : 327.95 seconds (486 problems not solved to optimality)
- SPRECHER : 1287.85 seconds (2062 problems not solved to optimality)

The New GRCPSP formulation of the SALBP is much more efficient than the Sprecher GRCPSP formulation, and this for any value of the parameters. In accordance with our conclusions derived earlier on the literature problem set, the dedicated branch-and-bound procedures, however, are more efficient than both GRCPSP formulations. Although SALB problems may be solved to optimality using resource-constrained project scheduling techniques, an optimal dedicated procedure performs much better.

Table 4 : SALBP CPU times with CI as network complexity measure

cycle time		10					20					30				
	tasks	10	15	20	25	30	10	15	20	25	30	10	15	20	25	30
CI																
	VH	.01	.03	.22	1.52		.01	.04	.16	.37		.01	.09	.79	4.09	
	FABLE	.01	.01	.03	.25		.01	.01	.01	.03		.01	.01	.01	.04	
0 - 3	EUREKA	.01	1.97	5.77	15.50		.00	.04	.09	1.03		.00	.00	.00	4.28	
	NEW	.01	.05	.57	119.96		.01	.06	.81	693.76		.01	.10	1.34	47.54	
	SPRECH.	.05	2.46	2704.45	—		.02	.38	1296.64	—		.02	.23	190.92	—	
	VH	.00	.03	.12	.81	20.44	.00	.03	.10	.35	2.36	.01	.05	.36	2.61	25.65
	FABLE	.00	.01	.02	.07	23.62	.01	.01	.01	.02	.10	.00	.01.	.01	.02	.02
4- 7	EUREKA	.00	1.21	3.97	7.57	10.32	.00	.01	.10	1.09	2.25	.00	.00	.00	.20	1.06
	NEW	.01	.04	.33	369.07	3339.54	.01	.04	.90	166.99	3600.00	.01	.08	.90	30.68	3600.00
	SPRECH.	.04	.60	1682.58	—	—	.02	.21	147.01	—	—	.01	.12	3.08	—	—
	VH		.02	.06	.55	4.81		.02	.05	.21	.69		.02	.09	.90	5.76
	FABLE		.01	.02	.06	1.96		.02	.02	.02	.03		.01	.02	.02	.02
8 - 11	EUREKA		.99	3.16	6.62	10.09		.00	.07	.55	1.04		.00	.00	.14	.83
	NEW		.02	.13	4.58	2180.17		.02	.17	6.25	2899.79		.05	.41	7.41	1881.16
	SPRECH.		.40	48.17	3579.54	—		.11	1.72	2986.13	—		.06	.78	2231.78	—
	VH			.04	.18	1.36			.03	.09	.32			.06	.20	1.35
	FABLE			.01	.03	.27			.01	.02	.03			.02	.02	.03
12 - 15	EUREKA			1.89	5.93	10.21			.02	.01	1.02			.01	.07	.65
	NEW			.10	1.26	839.04			.11	1.19	909.81			.31	2.53	172.57
	SPRECH.			6.53	2782.22	—			1.08	1204.03	—			.58	373.54	—
	VH				.12	.71				.06	.18				.15	.49
	FABLE				.02	.43				.04	.03				.02	.03
16 - 19	EUREKA				7.08	9.43				.00	1.45				.00	.96
	NEW				.58	181.89				.59	26.28				.94	24.60
	SPRECH.				1828.52	—				153.03	—				92.26	—

The results obtained indicate that FABLE is the most efficient dedicated procedure for solving the SALBP. Although the efficiency of FABLE and VH are somewhat comparable when the cycle time is small (for a cycle time equal to 10 is FABLE, on the average 10 % faster than VH), the efficiency gap becomes quite large (FABLE being almost three times faster) when the cycle time increases. Notice, however, that in practice, lines are generally operated at relatively small cycle times, because large cycle times go together with low output (rates). Given the fact that maximizing the output rate of a line is a common and relevant objective, it is important to have a line balancing procedure which performs well at relatively small cycle times.

Although the average CPU time required by EUREKA is higher than the average CPU time required by FABLE and VH, FABLE and VH manage to solve all problems optimally whereas EUREKA has to resort to a heuristic for 276 of the 6000 problems. Increasing the time window L for EUREKA will not only increase the number of problems solved to optimality, but also the average CPU time. Note that with a time window of 1800 seconds (so that a total of 3600 seconds may be spent on searching for the optimal solution,

which is the same as for the other procedures) EUREKA still does not find the (proven) optimal solution for 97 problems.

Table 5 : SALBP CPU times with F_{sub} as network complexity measure

Feasible Subsets	cycle time	10	20	30
	VH	.01	.01	.02
	FABLE	.01	.01	.01
0-100	EUREKA	.25	.00	.00
	NEW	.01	.02	.03
	SPRECHER	.12	.04	.03
	VH	.03	.03	.06
	FABLE	.02	.01	.01
101-200	EUREKA	1.95	.01	.00
	NEW	.06	.07	.16
	SPRECHER	2.67	.58	.28
	VH	.05	.05	.10
	FABLE	.02	.02	.01
201-300	EUREKA	2.80	.05	.00
	NEW	.13	.18	.34
	SPRECHER	94.95	9.74	5.05
	VH	.08	.07	.14
	FABLE	.02	.02	.02
301-400	EUREKA	4.67	.04	.00
	NEW	.29	.35	.71
	SPRECHER	666.28	69.00	30.83
	VH	.12	.09	.17
	FABLE	.03	.02	.02
401-500	EUREKA	5.70	.20	.01
	NEW	.74	.77	1.57
	SPRECHER	1692.93	608.57	193.33
	VH	.19	.11	.33
	FABLE	.03	.02	.02
501-1000	EUREKA	5.49	.25	.03
	NEW	2.07	1.58	2.52
	SPRECHER	2722.79	828.86	403.14
	VH	.49	.19	.68
	FABLE	.24	.02	.02
1001-2000	EUREKA	7.16	.71	.49
	NEW	23.58	10.98	13.84
	SPRECHER	3600.00	3214.87	2344.52
	VH	1.21	.30	1.57
	FABLE	.21	.02	.02
2001-5000	EUREKA	9.51	1.20	.59
	NEW	317.47	387.12	108.56
	SPRECHER	—	—	—
	VH	2.65	.59	3.14
	FABLE	.79	.02	.02
5001-10 000	EUREKA	9.74	1.02	.76
	NEW	1913.47	2312.36	1178.86
	SPRECHER	—	—	—
	VH	16.81	1.96	21.98
	FABLE	18.09	.08	.02
10 001-100 000	EUREKA	9.74	1.71	.92
	NEW	3199.76	3600.00	3301.43
	SPRECHER	—	—	—

Table 6 : SALBP CPU times with F'_{sub} as network complexity measure

Estimated Feasible Subsets	cycle time	10	20	30
0-100	VH	.01	.01	.01
	FABLE	.01	.01	.01
	EUREKA	.21	.00	.00
	NEW	.01	.01	.02
	SPRECHER	.09	.04	.03
101-200	VH	.03	.03	.05
	FABLE	.01	.01	.01
	EUREKA	1.66	.01	.00
	NEW	.04	.05	.11
	SPRECHER	1.26	.32	.17
201-300	VH	.04	.04	.08
	FABLE	.02	.01	.02
	EUREKA	2.33	.04	.01
	NEW	.09	.11	.24
	SPRECHER	5.00	1.02	.46
301-400	VH	.05	.05	.11
	FABLE	.01	.01	.02
	EUREKA	2.46	.03	.00
	NEW	.12	.18	.34
	SPRECHER	80.57	9.18	4.89
401-500	VH	.08	.06	.12
	FABLE	.03	.02	.02
	EUREKA	5.90	.03	.00
	NEW	.22	.30	.67
	SPRECHER	369.62	21.29	13.94
501-1000	VH	.12	.10	.24
	FABLE	.03	.02	.02
	EUREKA	4.24	.17	.02
	NEW	.59	.76	1.26
	SPRECHER	1811.14	313.62	113.91
1001-2000	VH	.25	.12	.47
	FABLE	.03	.02	.02
	EUREKA	6.17	.08	.04
	NEW	2.56	2.21	3.19
	SPRECHER	3292.85	1548.84	615.82
2001-5000	VH	.64	.22	.99
	FABLE	.21	.02	.02
	EUREKA	8.26	1.18	.49
	NEW	32.42	15.68	15.28
	SPRECHER	3570.41	3436.21	2994.81
5001-10 000	VH	1.21	.30	1.69
	FABLE	.23	.02	.02
	EUREKA	9.22	.98	.91
	NEW	486.78	557.10	164.53
	SPRECHER	—	—	—
10 001-200 000	VH	11.59	1.44	14.17
	FABLE	11.60	.06	.02
	EUREKA	9.98	1.45	.76
	NEW	2707.58	3140.04	2497.11
	SPRECHER	—	—	—

Table 7 : SALBP CPU times with F'_{seq} as network complexity measure

Feasible Sequences	cycle time	10	20	30
0-1000	VH	.01	.01	.01
	FABLE	.01	.01	.00
	EUREKA	.01	.01	.00
	NEW	.01	.01	.01
	SPRECHER	.06	.03	.02
1001-1000 000	VH	.02	.02	.04
	FABLE	.01	.01	.01
	EUREKA	.01	.01	.01
	NEW	.03	.03	.07
	SPRECHER	.46	.16	.10
1E6-1E9	VH	.05	.04	.08
	FABLE	.02	.02	.01
	EUREKA	.02	.01	.01
	NEW	.10	.12	.27
	SPRECHER	30.22	1.22	.57
1E9-1E12	VH	.15	.11	.37
	FABLE	.03	.02	.02
	EUREKA	.03	.02	.02
	NEW	.60	.65	1.40
	SPRECHER	2072.80	1734.44	311.37
1E12-1E15	VH	.51	.17	.64
	FABLE	.18	.02	.02
	EUREKA	.18	.02	.02
	NEW	16.52	11.09	11.79
	SPRECHER	3416.00	2827.24	2225.56
1E15-1E18	VH	1.10	.34	2.00
	FABLE	.18	.03	.02
	EUREKA	.18	.02	.02
	NEW	538.37	536.72	100.58
	SPRECHER	—	—	—
1E18-1E21	VH	4.59	.66	5.60
	FABLE	1.92	.03	.02
	EUREKA	1.92	.03	.02
	NEW	2065.48	2809.00	1775.07
	SPRECHER	—	—	—
1E21-1E24	VH	20.00	2.30	24.96
	FABLE	22.91	.10	.02
	EUREKA	22.91	.10	.02
	NEW	—	—	—
	SPRECHER	—	—	—

6.2.3. Complexity measures for the SALBP

6.2.3.1. General observations

Without going into issues of statistical significance, the results presented in Tables 2 to 7 allow the following observations to be made with respect to the computational effort required for solving a SALBP as a function of the tested measures of SALBP complexity :

- The larger *the number of tasks*, the harder the SALBP. This relationship holds for all solution methods included in the experiment.

- The effect of the *average number of tasks per station in the optimal solution* (or equivalently, the effect of the *cycle time*) differs among the solution procedures. For the VH procedure, the effect seems to be U-shaped. Problems with small or large cycle times are more difficult to solve to optimality than problems with an intermediate cycle time value. For the Sprecher GRCPSP formulation, an increase in the cycle time, or equivalently, in the average number of tasks per station in the optimal solution, results in an easier to solve SALBP. The same holds for for EUREKA and FABLE. This is in line with Johnson (1988) who reports that the performance of FABLE improves when the average number of tasks per station increases. The effect of the cycle time on the required CPU time for the New GRCPSP procedure, however, is not that clear. For problems with a small number of tasks (or low values of F_{sub} and F'_{seq}), the correlation between the cycle time and the required CPU time seems to be positive. However, for problems with a large number of tasks, the effect becomes U-shaped (for high values of CNC or relatively high values of F_{sub} and F'_{seq}), or reverse U-shaped (for low values of CNC or high values of F_{sub} and F'_{seq}). There is a strong indication that the relative efficiency of the tested SALBP-procedures critically depends on the value of the average number of stations in the optimal solution (cycle time). Since, as stated above, low cycle times are much more relevant in practice than high cycle times, solution procedures which perform well for problems with low cycle times have a competitive advantage.
- The larger the *coefficient of network complexity (CNC ; arcs/nodes)*, the easier the SALBP.
- The CPU time for solving the SALBP decreases as its *order strength (OS)* increases. This is consistent with earlier results (Johnson 1981, Baybars 1986).
- The larger the value of the *complexity index (CI)*, the easier to solve the SALBP.
- The larger the *number of feasible subsets* (F_{sub}), the larger the CPU time needed for solving the SALBP.
- The computational requirement for solving the SALBP increases with increasing *estimated number of feasible subsets* (the upper bound F'_{sub}).
- The larger the *estimated number of feasible sequences* (the upper bound F'_{seq}), the more difficult to solve the SALBP.

6.2.3.2. Statistical analysis

Inspired by these observations, we performed a statistical analysis in order to investigate the potential of the complexity measures for explaining variations in the required CPU time for solving SALBP-instances. We ran a *loglinear regression* (assuming an exponential effect of the complexity measures) and an *analysis of variance* (ANOVA; without specifying the form of the relationship), relating the required CPU time to the complexity measures. We did not include EUREKA in this analysis, since the efficiency of

EUREKA (as a heuristic solution procedure) should be expressed both by the required CPU time as well as the number of problems solved to optimality (or the average deviation). When all seven complexity measures are included in the regression model, the correlation between the complexity measures yield collinearity problems. Therefore, we cannot extract the effects of the individual measures from the regression (ANOVA) results and we have to look at the effects of the complexity measures individually (such that in each model, only one complexity measure is examined). Note, however, that all seven complexity measures can account a high percentage of the variability in the required CPU times :

- VH : 73% (ANOVA : 78%)
- FABLE : 31% (ANOVA : 34%)
- NEW GRCPSP : 90% (ANOVA : 95%)
- SPRECHER GRCPSP : 92% (ANOVA : 96%)

In our first analysis, we look at what the complexity measures tell us about the variations in the CPU times, when the effect of the number of tasks and the cycle time is already accounted for. Stated otherwise, we investigate what these measures explain beyond what is already explained by the number of tasks and the cycle time. The results of this analysis (using the *general linear models procedure* of *SAS*) are shown in Table 8. For each complexity measure, the table shows three results obtained for each solution procedure by both the regression model and ANOVA. For the regression results, a + *sign (- sign)* in the column with heading "effect", means that increasing values of the complexity measure have a positive (negative) effect on required CPU time. R^2 denotes the portion of the total variance in CPU times that the measure, in combination with the number of tasks and the cycle time, can account for. R^2_{part} denotes the extra explained variance beyond what could already be explained by the number of tasks and the cycle time. The higher the value of R^2_{part}, the more powerful the complexity measure is in explaining CPU time variations.

As expected, F_{sub} has the highest potential for explaining the variabilility in the CPU time required for solving the SALBP. However, the calculation of F_{sub} is NP-hard (Provan & Ball 1983), which puts a mortgage on its practical use as a complexity measure. According to expectations, the upper bound F'_{sub} does not perform as well. Also the upper bound on the number of feasible sequences, F'_{seq}, is not as good as F_{sub} in explaining variations in the SALBP CPU times. Among the network complexity measures (CNC, OS and CI), it appears form Table 8 that the order strength (OS) succeeds best in explaining variations in required CPU time, which makes it a very promising measure for predicting the CPU times needed by branch-and-bound procedures for solving the SALBP. However, F'_{sub}, which is a combined complexity measure of both the size and the network topology, performs equally well. Therefore, we can conclude that, when used in combination with the

number of tasks and the cycle time, both the OS and F'_{sub} are good measures of network complexity.

Table 8 : Effect of complexity measures on SALBP CPU times

		Proc.	VH			FABLE			NEW			SPRECHER		
Analysis	Complexity Measure		effect	R^2	R^2_{part}	effect	R^2	R^2_{part}	effect	R^2	R^2_{part}	effect	R^2	R^2_{part}
	CNC		-	66%	3%		30%	0%	-	85%	6%	-	90%	1%
R	OS		-	72%	9%		31%	1%	-	85%	6%	-	91%	2%
E	CI		-	70%	7%		30%	0%	-	88%	9%	-	90%	1%
G	Fsub		+	73%	10%		31%	1%	+	89%	10%	+	91%	2%
R	F'sub		+	72%	9%		30%	0%	+	88%	9%	+	91%	2%
.	F'seq		+	68%	5%		30%	0%	+	89%	10%	+	90%	1%
	CNC			69%	5%		31%	0%		89%	4%		93%	1%
A	OS			75%	11%		33%	2%		94%	9%		95%	3%
N	CI			72%	8%		32%	1%		93%	8%		94%	2%
O	Fsub			77%	13%		33%	2%		94%	9%		95%	3%
V	F'sub			75%	11%		32%	1%		93%	8%		95%	3%
A	F'seq			72%	8%		32%	1%		93%	8%		95%	3%

Note that for the New GRCPSP formulation, CI outperforms OS as a network complexity measure. This is consistent with the results of De Reyck and Herroelen (1995). In their study of complexity measures for the resource-constrained project scheduling problem (RCPSP), they confirm the explanatory power of the CI. Since the New GRCPSP formulation has much in common with the classic RCPSP, it is logical that the complexity measures for the RCPSP are also good measures of network complexity for the SALBP when modelled as a (G)RCPSP.

The fact that the effect of (network) complexity measures depends on the solution procedure used, illustrates - as already observed by Elmaghraby and Herroelen (1980) - that a measure of network complexity is indeed dependent on the *objective* of analysis and is 'necessarily' confounded by the *procedure* of analysis. The solution algorithm used to solve a problem is inextricably entwined with whichever properties we isolate and incorporate as a measure of complexity under the 'current state of technology'.

We also ran a loglinear regression (and an ANOVA) in order to examine the explanatory power of each of the complexity measures on their own, without including the number of tasks and the cycle time. The results of this analysis are represented in Table 9.

Table 9 : Individual effect of complexity measures on SALBP CPU times

Analysis	Complexity Measure	Proc.	VH			FAB			NEW			SPR		
			effect	sign.	R^2	effect	sign.	R^2	effect	sign.	R^2	effect	sign.	R^2
	TASKS		+	p<.001	61%	+	p<.001	27%	+	p<.001	79%	+	p<.001	87%
	CYCLE TIME		+	p<.05	2%	-	p<.05	3%		p>.250	0%	-	p<.05	2%
R	CNC			p>.250	0%	+	p<.05	2%		p>.250	0%	+	p<.05	1%
E	OS		-	p<.001	62%	-	p<.001	17%	-	p<.001	70%	-	p<.001	66%
G	CI		+	p<.001	18%	+	p<.001	15%	+	p<.001	23%	+	p<.001	40%
R	Fsub		+	p<.001	71%	+	p<.001	24%	+	p<.001	89%	+	p<.001	83%
.	F'sub		+	p<.001	70%	+	p<.001	19%	+	p<.001	88%	+	p<.001	86%
	F'seq		+	p<.001	65%	+	p<.001	24%	+	p<.001	88%	+	p<.001	84%
	TASKS			p<.001	61%		p<.001	28%		p<.001	85%		p<.001	90%
	CYCLE TIME			p<.001	3%		p<.001	4%		p>.250	0%		p<.001	2%
A	CNC			p<.001	12%		p<.001	4%		p<.001	8%		p<.001	10%
N	OS			p<.001	64%		p<.001	19%		p<.001	75%		p<.001	72%
O	CI			p<.001	30%		p<.001	18%		p<.001	32%		p<.001	48%
V	Fsub			p<.001	75%		p<.001	29%		p<.001	93%		p<.001	92%
A	F'sub			p<.001	73%		p<.001	25%		p<.001	92%		p<.001	93%
	F'seq			p<.001	68%		p<.001	28%		p<.001	93%		p<.001	93%

As expected, F_{sub}, F'_{sub} and F'_{seq} explain a large portion of the variations in the CPU time required for solving the SALBP. Surprisingly enough, the order strength (OS), which is a *network* complexity measure, can explain an unusual high percentage of CPU time variability. This confirms that the order strength is probably a good measure of network complexity for the SALBP. Note that the effect of the complexity index (CI) is positive, contrary to the previous results. This is due to the fact that this network complexity measure, in the absence of other (size related) complexity measures, tries to account for other variations besides network topology variations, such as the changing size and resource availability of the problem. The order strength, however, still has a negative effect.

From Tables 8 and 9, we can conclude that the best measures of SALBP complexity (next to the number of tasks and the cycle time), in order of explanatory power, are :

- F_{sub}
- OS and F'_{sub}
- F'_{seq}
- CI
- CNC.

Since F_{sub} is very hard to calculate and therefore not very useful as a complexity measure, the best combination of complexity measures is the number of tasks (as a *size* measure), the cycle time (as a *resource availability* measure) and the order strength or F'_{sub} (as *network complexity* measures). When generating problem instances, however, the order

strength is much easier to manipulate than F'_{sub}. This brakes a lance for a composite measure *number of tasks + cycle time + OS*. F_{sub} (and its estimator F'_{sub}) can be used for an ex-post description and validation of a generated benchmark problem set.

Our suggested composite measure is in line with previous experimental investigations described in the literature (Johnson 1988, Baybars 1986). Our reassuring findings validate their way of generating problem instances while controlling the number of tasks, the cycle time (as an indication of the average number of tasks per station in the optimal solution) and the order strength.

7. Conclusions

In this paper we exploit the similarity between the simple assembly line balancing problem (SALBP) and the generalized resource-constrained project scheduling problem (GRCPSP) by studying two GRCPSP-based formulations for the SALBP. Both formulations assume a single renewable resource. The first formulation (*Sprecher*) assumes finish-start precedence constraints, unit resource requirements and varying resource availability. The second formulation (*New*) assumes start-start precedence relations, unit activity durations and constant resource availability. Computational results obtained on two problem sets (a small set derived from the literature and a new test battery consisting of 6000 instances) indicate that *New* outperforms *Sprecher*. An extensive validation experiment on the same problem sets indicates that, despite of the strong similarities among the SALBP and the GRCPSP, the GRCPSP-based solution procedures as they stand, are outperformed by the dedicated line balancing procedures *VH*, *FABLE* and *EUREKA*. While EUREKA seems to outperform VH and FABLE on the problems taken from the literature, this is no longer the case on the new set of 6000 problem instances generated using the network generator PROGEN. On this set, FABLE is clearly the most efficient procedure. Although the efficiency of FABLE and VH are comparable on problem instances with relatively small cycle times (i.e., on problem instances with high line throughput), the efficiency gap becomes quite large as the cycle time goes up.

Extensive computational tests supported by statistical analysis confirm that the use of a composite measure of complexity based on the number of tasks, the cycle time and the order strength has enough potential for explaining the variability in the CPU time required for solving SALBP instances. This reassuring conclusion is in line with common practice in validating solution procedures for the SALBP.

Acknowledgement

We would like to thank Andreas Drexl, Rainer Kolisch and Arno Sprecher for providing us with the project generator PROGEN. We also thank Erik Demeulemeester, Thomas Hoffmann and Roger Johnson for providing us with the code of the GRCPSP, EUREKA and FABLE respectively.

Appendix: The complexity index (CI)

Let D = (N, A) be a two-terminal *AoA network*, where N = {1, 2, ..., n} is the set of nodes representing the project events, and A is the set of arcs representing network activities. We assume, without loss of generality, that there is a single start node 1 and a single terminal node n, $n = |N|$. Since D is acyclic, we assume that its nodes are topologically numbered; i.e., $i \prec j$ whenever there exists an arc joining i to j. The resulting graph will be referred to as a *st-dag*.

Bein et al. (1992) define complexity in terms of a sequence of node reductions of a st-dag. There are three types of reductions : parallel, series and node reductions. These reductions, when applied consecutively in the right order, can reduce any dag to one single arc. A *parallel reduction* at i, j replaces two or more arcs $a_1, a_2, \ldots a_k$, all joining i to j, by a single arc $a = (i, j)$. A *series reduction* at i is possible when $a = (i, j)$ is the unique arc into j and $b = (j, k)$ is the unique arc out of j : a and b are replaced by a single arc $c = (i, k)$. Let [D] denote the network obtained by applying to D all series-parallel arc reductions. If D = [D] then D is said to be *irreducible*.

Following Bein et al. (1992), we say that node j of an irreducible network is eligible for a *node reduction* when j has unit in-degree or out-degree, and $j \neq 1, n$. Let $a = (i, j)$ be the unique arc into j and $b_1=(j, k_1),\ldots, b_s=(j, k_s)$ be the arcs out of j. Then the reduction of node j replaces $a, b_1,\ldots, b_s$ by the arcs $c_1 = (i, k_1),\ldots c_s = (i, k_s)$. The case where j has unit out-degree is symmetric. Note that in an irreducible network any node whose only predecessor is 1 or whose only successor is n is eligible for reduction. Therefore, every acyclic network can be reduced to the single arc (1, n) by a sequence of node reductions interleaved with series and parallel reductions. The number of node reductions in such a sequence may differ. Bein et al. (1992) define the reduction complexity of D as the minimum number of node reductions sufficient (along with series and parallel reductions) to reduce D to a single arc. More formally, let $D \circ j$ denote the network obtained from reduction of node j in D. Then the reduction complexity is the smallest q for which there exists a sequence of nodes $(j_1, j_2,\ldots, j_q)$ such that $[\ldots[[[D] \circ j_1] \circ j_2]\ldots \circ j_q] = (1, n)$. Such a sequence is called a *reduction sequence*. The length of a reduction sequence; i.e., the reduction complexity, is taken as our definition of the *complexity index, CI,* of D. Since all series-parallel networks have a CI-value equal to zero, the complexity index CI seems to be a good measure of how close the network is to being series-parallel. Bein et al. (1992) have developed an algorithm for calculating the CI of a dag in polynomial time. A more detailed description of the algorithm and computational results can be found in De Reyck and Herroelen (1995).

Because the CI is defined only for AoA networks and the SALBP is formulated as an AoN network, we cannot simply calculate the CI of the 6000 networks generated by PROGEN. However, Kamburowski et al. (1992, 1993) have developed a polynomial algorithm which transforms an AoN network in an equivalent AoA network with minimal CI and the minimum number of nodes. A detailed description of this algorithm and computational experience can be found in De Reyck and Herroelen (1995). Consequently, when calculating the CI of a given AoN network, we have to transform it to an equivalent AoA network using the algorithm of Kamburowski et al. (1992, 1993), and then calculate the CI of the AoA network using the algorithm of Bein et al. (1992).

References

BAYBARS, I., 1986, A survey of exact algorithms for the simple assembly line balancing problem, *Management Science*, **32**(8), 909-932.

BEIN, W. W., KAMBUROWSKI, J. and STALLMANN, M. F. M., 1992, Optimal reduction of two-terminal directed acyclic graphs, *SIAM Journal on Computing*, **21**, 1112-1129.

BLACK, O. J., 1965, An algorithm for resource leveling in project networks, *Unpublished Report, Department of industrial Administration, Yale University, New Haven Connecticut*, U.S.A.

BLAZEWICZ, J., LENSTRA, J. K. and RINNOOY KAN, A. H. G., 1983, Scheduling projects to resource constraints - Classification and complexity, *Discrete Applied Mathematics*, **5**, 11-24.

BOWMAN, E. H., 1959, The schedule-sequencing problem, *Operations Research*, **7**, 621-624.

DAR-EL, E. M., 1973, MALB-A heuristic technique for balancing large single-model assembly lines, *AIIE Transactions*, **5**, 343-356.

DAVIS, E. W. and HEIDORN, G. E., 1971, An algorithm for optimal project scheduling under multiple resource constraints, *Management Science*, **17**(12), B-803-B-816.

DAVIS, E. W., 1966, Resource allocation in project network models - A survey, *Journal of Industrial Engineering*, **17**(4), 177-188.

DAVIS, E. W., 1973, Project scheduling under resource constraints - Historical review and categorization of procedures, *AIIE Transactions*, **5**(4), 297-313.

DAVIES, E. M., 1974, An experimental investigation of resource allocation in multiactivity projects, *Operational Research Quarterly*, **24**, 587-591.

DEMEULEMEESTER, E. L. and HERROELEN, W. S. 1994, A branch-and-bound procedure for the generalized resource-constrained project scheduling problem, *Operations Research*, to appear.

DE REYCK, B. and HERROELEN, W. S., 1995, On the use of the complexity index as a measure of complexity in activity networks, *European Journal of Operational Research*, to appear.

DREXL, A., 1990, Fliessbandaustaktung, Maschinenbelegung und Kapazitätsplanung in Netzwerken - Ein integrierender Ansatz, *Zeitschrift für Betriebswirtschaft*, **60**(1), 53-70.

DREXL, A., 1994, private communication.

ELMAGHRABY, S. E., 1977, *Activity Networks - Project Planning and Control by Network Models*, Wiley Interscience, New York.

ELMAGHRABY, S. E. and HERROELEN, W. S., 1980, On the measurement of complexity in activity networks, *European Journal of Operational Research*, **5**, 223-234.

GUTJAHR, A. L. and NEMHAUSER, G. L., 1964, An algorithm for the line balancing problem, *Management Science*, **11**(2), 308-315.

HELD, M., KARP, R. M., and SHARESHIAN, R., 1963, Assembly line balancing - dynamic programming with precedence constraints, *Operations Research*, **11**(3), 442-459.

HERROELEN, W. S., 1972, Resource-constrained project scheduling - The state of the art, *Operational Research Quarterly*, 23, 261-275.

HERROELEN, W. S., 1980, Assembly line balancing problems in perspective, *Tijdschrift voor Economie en Management*, **XXV**(1), 61-76.

HERROELEN, W. S. and DEMEULEMEESTER, E. L., 1994, Recent advances in branch-and-bound procedures for resource-constrained project scheduling problems, Chapter 12 in *Scheduling Theory and Its Applications*, Ph. Chrétienne, E. G. Coffman Jr., J. K. Lenstra and Z. Liu (eds.), John Wiley & Sons.

HOFFMANN, T. R., 1992, Eureka : A hybrid system for assembly line balancing, *Management Science*, **38**(1), 39-47.

ICMELI, O., 1993, Project scheduling problems: A survey, *International Journal of Operations and Production Management*, **13**(11), 80-91.

IGNALL, E. J., 1965, A review of assembly line balancing, *Journal of Industrial Engineering*, **XV**(4), 244-254.

JACKSON, J. R., 1956, A computing procedure for the line balancing problem, *Management Science*, **2**(3), 261-271.

JOHNSON, R. V., 1981, Assembly line balancing algorithms : Computation comparisons, *International Journal of Production Research* , **19**, 277-287.

JOHNSON, R. V., 1988, Optimally balancing large assembly lines with 'FABLE', *Management Science*, **34**(2), 240-253.

JOHNSON, R. V., 1993, Note: Microcomputer performance of "FABLE" on Hoffmann's data set, *Management Science*, **39**(10), 1190-1192.

KAMBUROWSKI, J., MICHAEL, D. J. and STALLMANN, M., 1992, Optimal construction of project activity networks, *Proceedings of the 1992 Annual Meeting of the Decision Sciences Institute*, San Francisco, 1424-1426.

KAMBUROWSKI, J., MICHAEL, D. J. and STALLMANN, M., 1993, On the minimum dummy arc problem, *Revue Française de Recherche Opérationelle* , **27**, 153-168.

KAO, E. P. C. and QUEYRANNE, M., 1982, On dynamic programming methods for assembly line balancing, *Operations Research*, **30**, 375-390.

KOLISCH, R., SPRECHER, A. and DREXL, A. (1992), Characterization and generationof a general class of resource-constrained project scheduling problems: Easy and hard instances, *Management Scienc* e, to appear.

MANDEVILLE, D. W., 1965, The development of network analysis resource balancing methods from assembly line balancing techniques, *Unpublished MS thesis, Purdue University, Lafayette*, Indiana.

MASTOR, A. A., 1970, An experimental and comparative evaluation of production line balancing techniques, *Management Science*, **16**(11), 728-746.

MOODIE, C. L. and MANDEVILLE, D. E., 1966, Project resource balancing by assembly line balancing techniques, *Journal of Industrial Engineering*, *XVII*(7), 377-383.

PASCOE, T. L., 1966, Allocation of resources - CPM, *Revue Française de Recherche Opérationelle*, **38**, 31-38.

PATTERSON, J. H. and ALBRACHT, J. J., 1975, Assembly line balancing : 0-1 programming with Fibonacci search, *Operations Research,* **23**, 166-174.

PRITSKER, A. A. B., WATTERS, W. D. and WOLFE, P. M., 1969, Multiproject scheduling with limited resources: A zero-one programming approach, *Management Science*, **16**(1), 93-108.

PROVAN, J. S. and BALL, M. O., 1983, The complexity of computing cuts and of computing the probability that a graph is connected, *SIAM Journal of Computing*, **12**, 777-788.

SALVESON, M. E., 1955, The assembly line balancing problem, *Journal of Industrial Engineering*, **6**, 18-25.

SCHRAGE, L. and BAKER, K. R., 1978, Dynamic programming solution of sequencing problems with precedence constraints, *Operations Research*, **26**, 444-449.

SPAAS, Ph. and HERROELEN, W. S., 1992, Decision support software for assembly line balancing, *Bedrijfeconomische Verhandeling N° 9203, Department of Applied Economics, Katholieke Universiteit Leuven*, Belgium.

SPARLING, D. and MILTENBURG, J., 1994, The U-line balancing problem - Multiple U-lines, *ORSA/TIMS Meeting Detroit*, October 23-26, 1994.

SPRECHER, A., 1994, *Resource-constrained Project Scheduling - Exact Methods for the Multi-mode Case*, Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin.

TALBOT, F. B. and PATTERSON, J. H., 1984, An integer programming algorithm with network cuts for solving the single model assembly line balancing problem, *Management Science*, **30**, 85-99.

TALBOT, F. B., PATTERSON, J. H., and GEHRLEIN, W. V., 1986, A comparative evaluation of heuristic line balancing techniques, *Management Science*, **32**(4), 430-454.

THANGAVELU, S. R. and SHETTY, C. M., 1971, Assembly line balancing by zero-one integer programming, *AIIE Transactions*, **3**, 61-68.

VAN ASSCHE, F. and HERROELEN, W. S., 1978, An optimal procedure for the single-model deterministic assembly line balancing problem, *European Journal of Operational Research*, **3**(2), 142-149.

WEE, T. S. and MAGAZINE, M. J., 1982, Assembly line balancing as generalized bin packing, *Operations Research Letters*, **1**, 56-58.

WHITE, W. W., 1961, Comments on a paper by Bowman, *Operations Research*, 9, 274-276.

WIEST, J. D., 1963, The scheduling of large projects with limited resources, *Unpublished PhD thesis, Carnegie Institute of Technology* .

WILSON, R. C., 1964, Assembly line balancing and resource leveling, *University of Michigan Summer Conference, Production and Inventory Control* .